M7011E - Design of Dynamic Web Systems
Project report

Authors
Peder Johansson pedjoh-7@student.ltu.se
Chonratid Pangdee chopan-7@student.ltu.se

January 2021

Contents

1 Introduction

In this project we took on the role as a web developing company who had been tasked to create a web system for controlling the production and consummation of electricity for several households in a small scaled market. We were also asked to create a simulator that was supposed to simulate wind as well as the electricity that was produced and consumed by users.

1.1 System introduction

The overall system (figure 1) consist of three major components, a **system**, a **client** (GUI) and, an **API**, each one with its own sub components. It operates on a web-server running NodeJS, which is a run-time environment that enables running JavaScript from the server-side. The System- and the API component operates at the the back-end of the server and serves as a web-service for the application. The client (GUI) component operates at the front-end server, which is running on React JavaScript library, and serves as an end-user interface for controlling the simulation. All data of the simulation are stored in a database using SQLite database management system. Other libraries worth mentioning that were used on the client system are Axios [3], for handling the communication between the client component and the API, and Bootstrap [6] for the design of the web pages.

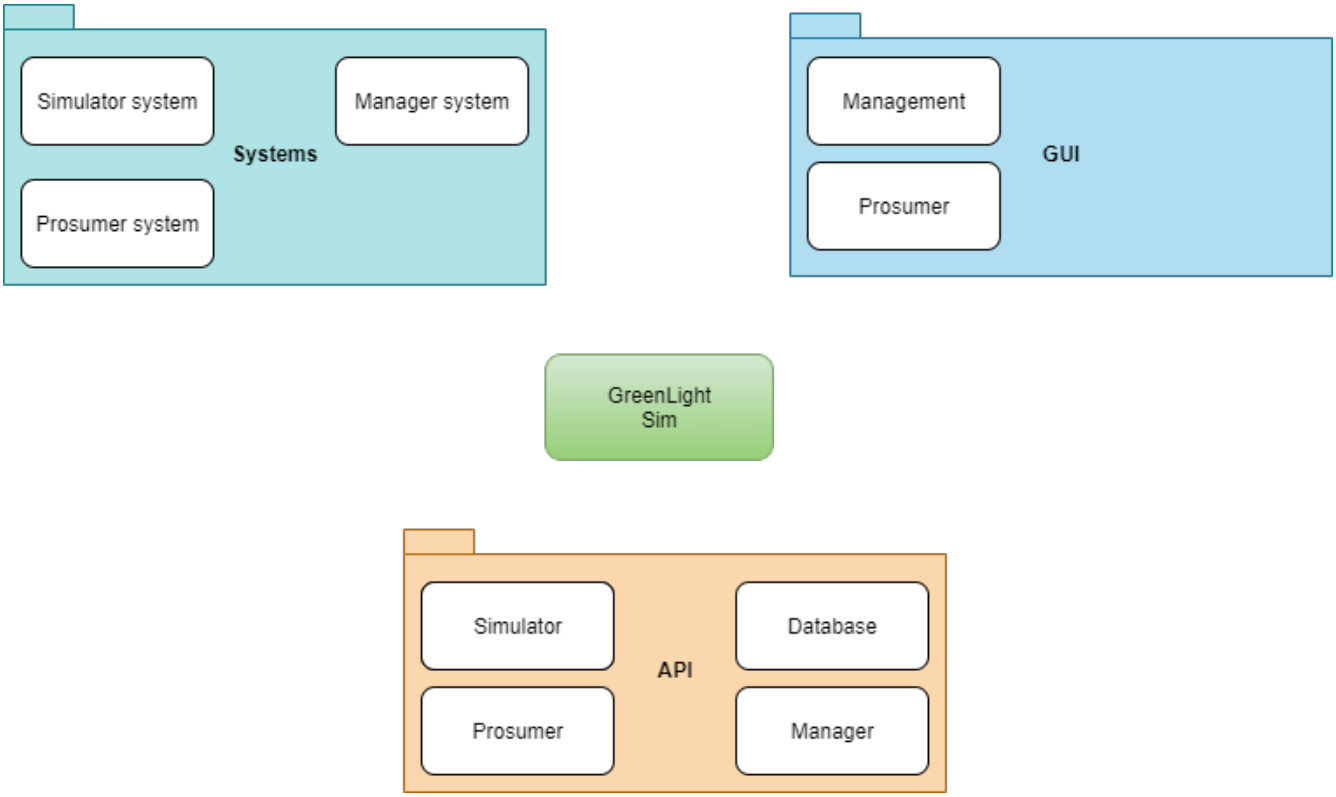

Figure 1: The overall system components.

1.1.1 The system component

The system component (figure 2) serves as the main component of the system. The purpose of this component is to simulate the whole system. Each sub component handles separate simulation tasks and are communicating with each other via the API interface.

Simulator system: The simulator system simulates the parameters needed for the simulation. The parameters are the weather (wind), the electricity consumption, and the pricing of the electricity. The model for calculating the weather (wind) is based on real historic values gathered from SMHI and Luleå Airport wind statistics [1] for the year 2019. The model for simulation of the electricity consumption is based the report by Pombeiro1 et al. [7]. The calculation of pricing is based on the market demand and supply which are gathered from the database.

Prosumer system: The prosumer system simulates the electricity production of the wind turbine installed at each prosumer household, and the electricity consumption of the prosumer household using the values obtained from the Simulator system via the simulator API. It also controls the amount of electricity the household sells to the market and store to the household buffer (battery) using parameters set by the end-user in the client (GUI) component.

Manager system: The manager system simulates a coal plant that produces electricity to the grid/market. It holds a buffer that all prosumers can contribute to by selling portions of their net-production. The main functionalities of the manager system is to control the electricity grid by using parameters set by a manager in the client component.

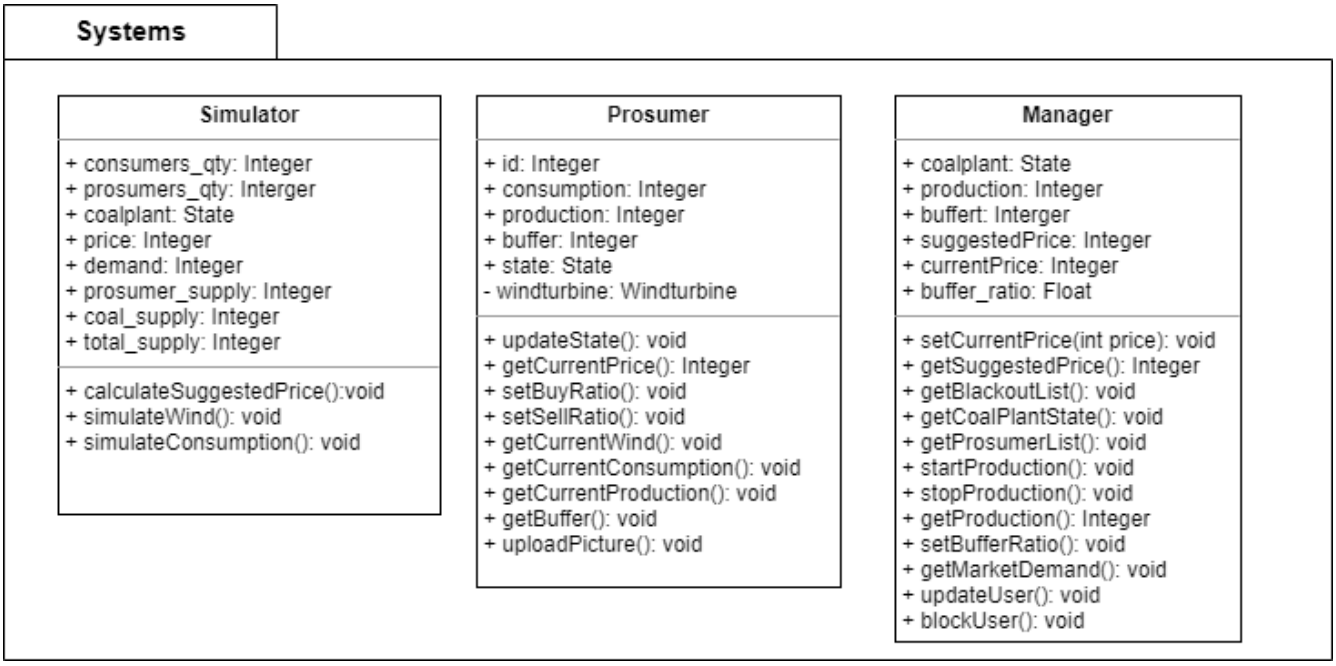

Figure 2: The three sub-components of the System component.

1.1.2 The client (GUI) component

The Client component is the main component for controlling the displaying the simulation. Figure 3 shows a rough prototype sketch of the sub-components of the GUI. It consist of a **registration page**, where prosumers and managers can sign up to get access to the simulation. A **prosumer page** where the prosumers can view their system values like current wind speed, current production and consumption, and control the parameters of the simulation for their household such as buy- and sell ratio wich dictates the amount of the net production that are sold to the market. A **manager page** where the manager/s can view and control the current state and parameters of the coal plant. The manager/s also has an overview of the whole grid and all the prosumers connected to the grid. Some of the functionalities accessible by the manager are view information of every prosumer household variables, blocking prosumer from selling electricity to the market, and deleting prosumer from the grid.

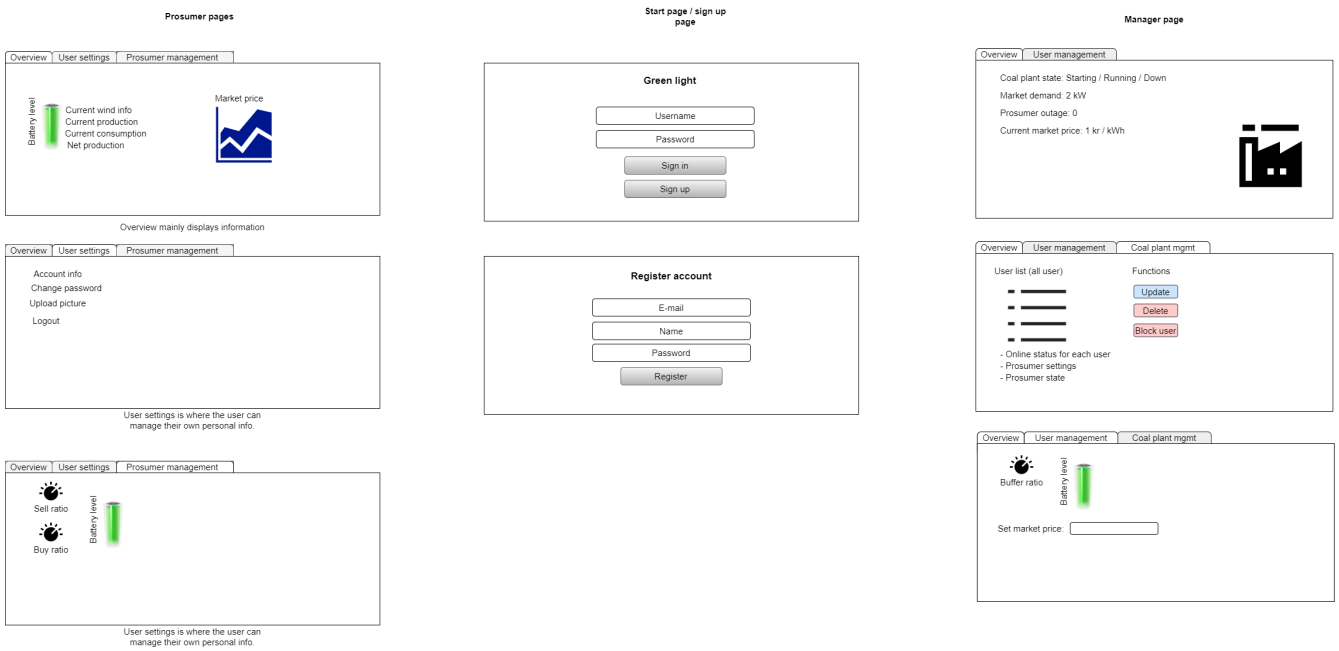

Figure 3: The prototype of the client GUI.

1.1.3 The API component

The API component serves as a main interface for communication between the different components of the system. This component is build on GraphQL, which is a open-source query and manipulation language that is used for API:s. It consist of three different endpoints, one for each sub-system in the system component. Each endpoint has its own set of queries for passing and gathering data from the API. The security mechanism used for accessing the API is based on JSON Web Tokens, which is library used for creating access tokens to certify user identity. We have used JWT to create access tokens for users to verify that they have access to certain data. The API component has a direct communication with the system component. The system component in turns has a direct connection to the database component. Some of the accessible data from the database is shown in the figure 4.

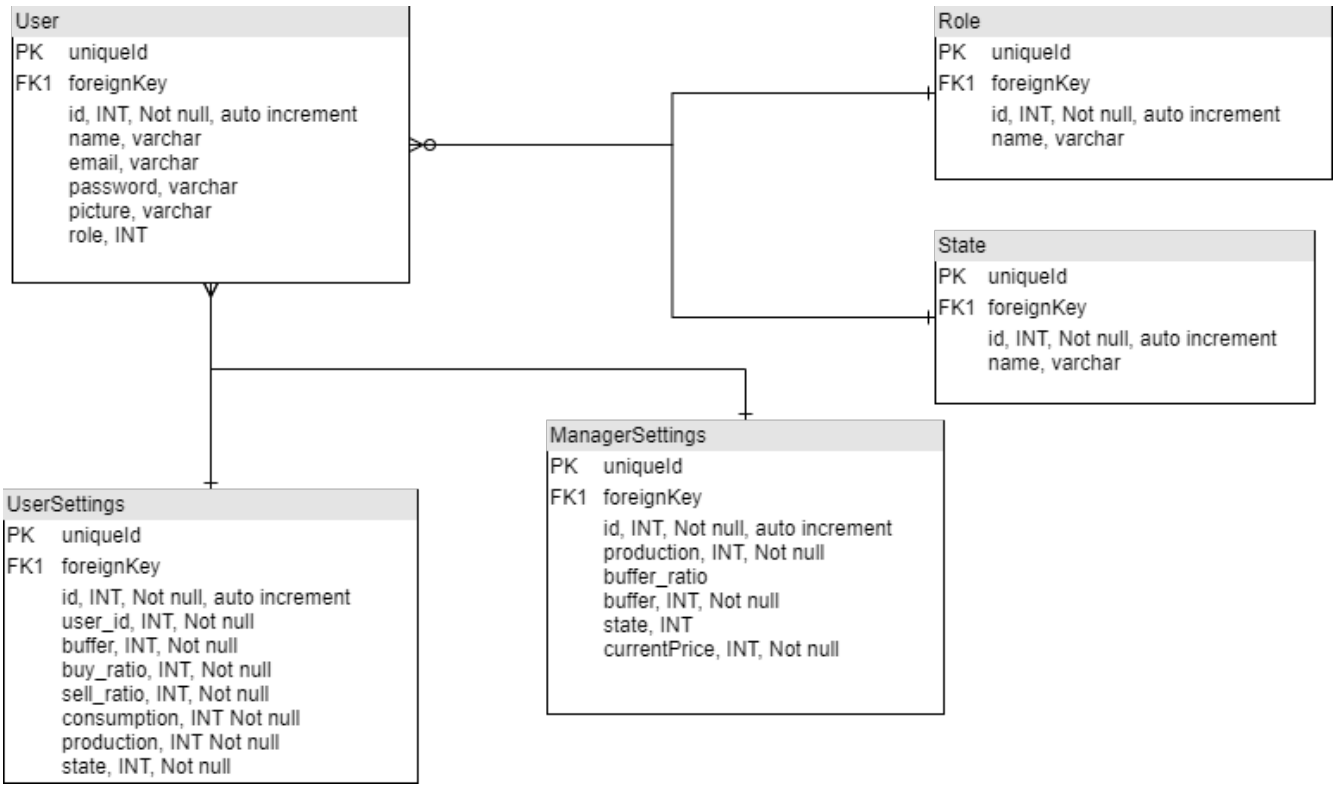

Figure 4: The database schema.

2 Method

2.1 Design choices

When creating the project we used modular programming. We wanted to create the different modules first and later on linking them together. Creating the project using this method has lots of benefits. It allow project members to work independently of each other. This is great for speeding up development since group members can put a bigger focus on working on parts were they have previous knowledge. When thinking of the learning aspect this method have had the downside of not letting members gain as much new knowledge as possible. Using modular programming simplifies further development since the different parts of the program are not heavily dependent on each other. This means that it becomes easier to update certain parts of the project with out ruining the functionality of the other parts.

When working on the individual parts of the code we had the idea of loose coupling in mind. We wanted different parts work independently, the reasoning was basically the same as for why we used modular programming. We wanted it to be easy to update code without loosing functionality in other functions or classes.

2.2 Scalability analysis

At the beginning of the project we were choosing between using graphQL or REST in the end we decided to use graphQL. GraphQL is able to retrieve data more efficiently than REST since graphQL enables us to fetch exactly the data we want with a single request, REST tend to have the problem that responses often have to much or to little data. Another advantage of grahphQL is that we are able to collect data from multiple sources using a single request. A disadvantage with graphQL is that it's hard to handle complex query. We did not encounter this problem, but it could become a problem if someone were to expand on this project and add more complexity to the queries.

We believe we may face issues regarding scalability because of the usage of SQLite since it is not designed for scalability. We would most likely get issues if we were to get a lot of traffic on our web application. To improve scalibaility one idea would be to change database to a more scaleable one.

2.3 Security analysis

One of the main security features we implemented is access tokens that we save in cookies. Users get a access token in order for the server to verify the user and only giving that user information that they are meant to have access to. We also implemented password encryption to strengthen the security for the users. One security flaw we have is that if someone were to get access to a users access token while they are still logged in that person would be able to gain access to the users information.

3 Result and discussion

3.1 The final product

The final product is a complete simulation system with basic features and functionalities described in the project problem description with a somewhat simple GUI in the sense of the look and feel. With a little more time, we could implement more advance features like introducing temperature as a parameter in the simulator component or introducing GPS coordinates as a parameter, which will affect the simulation of the weather.

3.1.1 Demo

A working system is deployed to the Ludd VPS server and is accessible from `http://130.240.200.52:3000/`. A small set of Prosumers and Managers are already in the system and system can be tested with these credentials:

Role	email	password
Prosumer	d1@greenlight.org	d1
Manager	manager01@greenlight.org	manager01

It is also possible to register a new prosumer or manager using the registration form accessible from the login page for each role. As a safety mechanism, to ensure that only authorized managers can sign up on the page, there is an input field in the registration form labelled "Confirm received key", that takes the sting "manager" for manager registration, and "prosumer" for prosumer registration. In later deployment, the purpose for this safety is to use a token that would be received via e-mail.

3.2 Challenges

A lot of the challenges we faced during the project was regarding lack of knowledge. We were both new to graphQL, JavaScript and React this lead to a lot of minor problems. This resulted in that many things took more time than we initially thought they would take. The backend is able the handle more than we were able to implement in the frontend. We underestimated the time it would take to get frontend working as intended. The main cause for problems has been time management and estimating how long it would take to do certain implementations. This is quite unfortunate since we had intended to implement more features than the basic ones that were required but in the end were not able to do this due to time restraints.

3.3 How to separate the simulation more from the API

In the current implementation, we have an instance of simulation for each part in each API endpoint. This could be separated by moving the initialization of the simulation part into a separate process and create a new API endpoint that could be more generally implemented linked to that process. We realized that our implementation is not optimal at later stage of the project and we decided to keep it this way.

3.4 Future work

Some future work that would not be hard to implement is other weather features than wind. We have a json file containing lot of data of different weather features. We used real wind data to create our random wind data for our simulator (the reason being to create realistic weather conditions). This can be done with the other weather data we have. The functions we use for the wind could quite easily be changed to be able to handle other types of information. If we had more time we would make a cleaner design for our frontend. We waited until the end to create our frontend, and because of the lack of time we had left until the deadline we had to prioritise getting the wanted functionality over a better looking design.

Napkinsketches

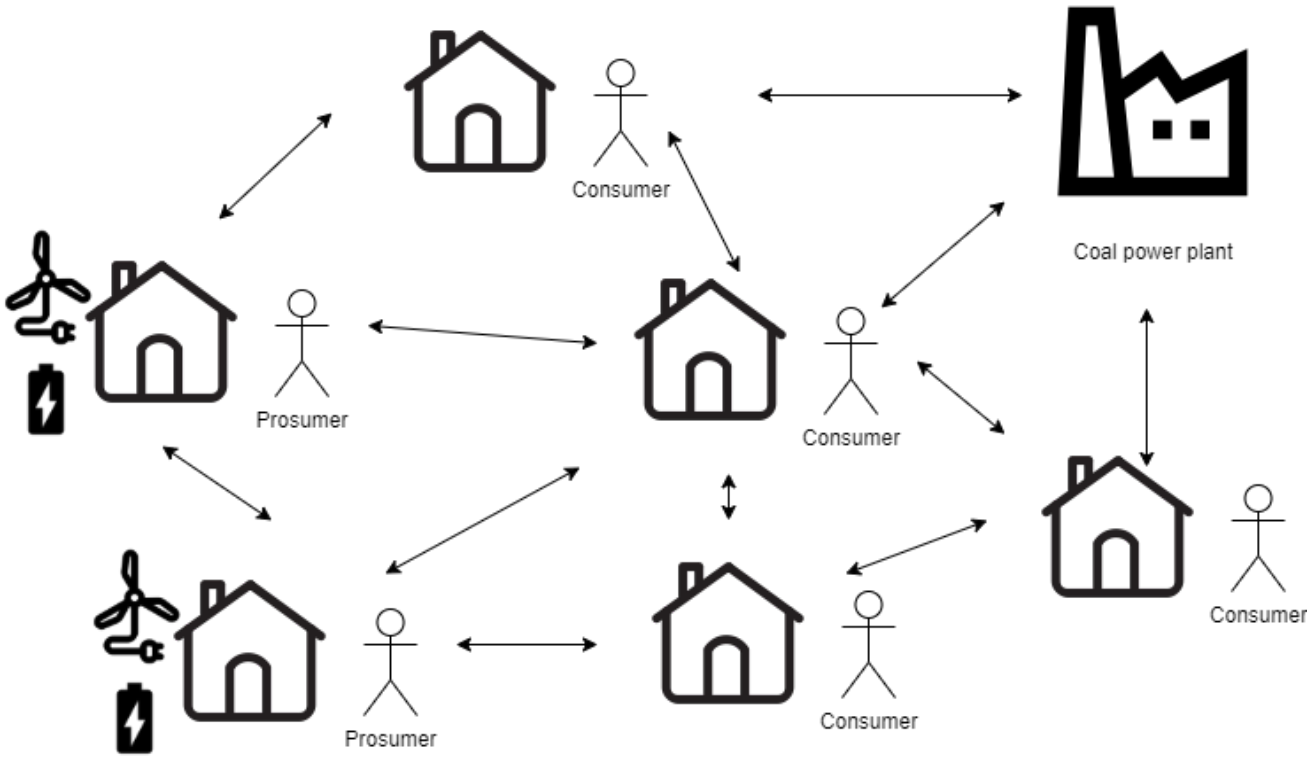

Figure 5: Napkinsketch of the simulation.

References

[1] SMHI wind statistics
https://www.smhi.se/data/meteorologi/vind

[2] React webpage
https://reactjs.org/tutorial/tutorial.html

[3] Axios
https://www.npmjs.com/package/axios

[4] JSON Web Token
https://jwt.io/introduction/

[5] W3Schools
https://www.w3schools.com/js/DEFAULT.asp

[6] React-Bootstrap
https://react-bootstrap.github.io/getting-started/introduction/

[7] Henrique Pombeiro1, André Pina1, Carlos Silva: *Analyzing Residential Electricity Consumption Patterns Based on Consumer's Segmentation*
http://ceur-ws.org/Vol-923/paper05.pdf

A Time analysis

A the beginning, the majority of the time was spent on planning the structure and the design of the system, and gathering information of different tools that are available for building the system. Then we spent time on setting up getting familiar with the tools and the environment. Once we have decided on the structure and the design we started the implementation. At first we were a bit concerned that we had put to much time into planing, but as we continued working on the project we realised that planning saved us an enormous amount of time. One example of this was when we created our simulator. When we starting writing code we started with working on the simulator since it would be used to get information in the other parts of the project. We had already decided on what functions and other code was need for the different parts of the simulator before we started on it. This made it easy to pick something that was needed to be done and inform the group member what was already being worked on without any confusion that could lead to us working on similar things.

B Contributions to the project

Since our project group is small, we started with pair programming were we initiated the project and build the base component of the system together. After that, we took an agile approach and started to work in sprints where we implemented the functionalities and features one by one from the planning. Although we divide the task and implementation for each project member, we spent most of the development time together, which has enabled us to work more closely and discuss problems and solutions in a more effective way.

C Goal

The main goal of the project was to meet all the requirements for basic functionalities and features stated in the problem description of the project. We feel that we have met that goal and should receive at least grade 3.

D Github link:

`https://github.com/chopan-7/M7011E_project`